

A Rewriting System for the Assessment of XACML Policies Similarity

M. Mejri¹ and H. Yahyaoui²

¹Computer Science Department, Laval University, Canada

mejri@ift.ulaval.ca

²Computer Science Department, Kuwait University, Kuwait

hamdi@cs.ku.edu.kw

Abstract

We propose in this paper a policies similarity approach which is performed in three steps. The first step is concerned with the formalization of a XACML policy as a term in a boolean algebra while taking into account the policy and rule combining algorithms. This formalization is based on Security Policy Language (SePL) which was proposed in a previous work. In the second step, the SePL term is transformed into a term in a boolean ring. In the third step, the two policy terms, which are derived from the previous step, are the input to a rewriting system to conclude which kind of relation exists between these security policies such as equivalence, restriction, inclusion, and divergence. We provide a case study of our approach based on real XACML policies and also an empirical evaluation of its performance.

Keywords: Security Policies; Similarity; Rewriting; XACML.

1. Introduction

Standard security policy languages are designed for the sake of creating a common background for providers to define their security constraints. A paramount benefit from such standardization is interoperability between different systems. Generally, the proposed security languages come with an informal syntax and semantics, which makes the learning curve of such languages high. One of these languages is eXtensible Access Control Markup Language (XACML) [20]. It is a standard XML based language that can be used in the definition of Web service policies. XACML comes with a set of combining algorithms that allow the integration of different security policies. This language is not endowed with a formal semantics and its combining algorithms are not formalized. Therefore, such famous standard language lacks a formal basis that allows to reason about security policies. For instance, the consistency of security policies can be established. One of the critical issue in reasoning about security policies is the similarity between security policies. Often, we need to know if a certain security policy is equivalent or not to another one to be sure that we don't have any conflict between policies. Such assessment can be done only if a rigorous formal framework is designed to capture the informal semantics of security policy languages such as XACML. Accordingly, we provided

in a previous work [17] a fully-fledge, compact and formal language called the Security Policy Language (SePL) to define formally security policies. The syntax of SePL includes several operators for the integration of policies and it is endowed with a denotational semantics that is a generic semantics, i.e., which is independent of any evaluation environment. We proved the completeness of SePL with respect to sets theory. Based on the elaborated semantics, we proposed a formalization of a subset of XACML 3.0. Furthermore, we provided a semantics for XACML policy combining algorithms. We build in this paper on top of the elaborated formal framework for security policies to reason about the similarity of security policies. Our policies similarity approach is performed in three steps. The first step is concerned with the formalization of a XACML policy as a term in a boolean algebra while taking into account the policy and rule combining algorithms. This formalization is based on SePL. In the second step, the SePL term is transformed into a term in a boolean ring. In the third step, the two policy terms, derived from the previous step, will be the input to a rewriting system to conclude which kind of relation exists between these security policies such as equivalence, restriction, inclusion, and divergence. We provide a case study of our approach on real XACML policies and also an empirical evaluation of its performance. The contributions of this work are as follows:

- The design of a transformation of XACML policies into semantic terms in a boolean ring.
- The proposal of a rewriting system for security policies that allows to establish the similarity between their semantic terms.
- The evaluation of the proposed technique.

The rest of the paper is divided as follows. In Section 2, we discuss the related work regarding policies similarity. Section 3 is dedicated to the presentation of the proposed approach. In Section 4, we present the mapping of XACML policies into terms of a boolean ring. Section 5 is devoted to the presentation of the proposed similarity algorithm. In section 6, we provide an experimental evaluation of the similarity algorithm. Some concluding remarks and future work are provided in Section 7. All the details regarding SePL syntax and semantics are provided in the Appendix.

2. Related Work

Policies integration is a paramount issue for service providers. An important step in such integration is to analyze the similarity of the policies. Unfortunately, this problem was not deeply investigated and only a few approaches were proposed. We review in what follows these initiatives and pinpoint their contributions and limitations.

Exam [15] is a complete environment for the analysis and management of access control policies. It supports the acquisition, search, analysis of similarity, and integration of security policies. This method uses the Multiple Terminal Binary Decision Diagram (MTBDD) as a representation of a policy. The MTBDD of a security policy is an acyclic directed graph whose

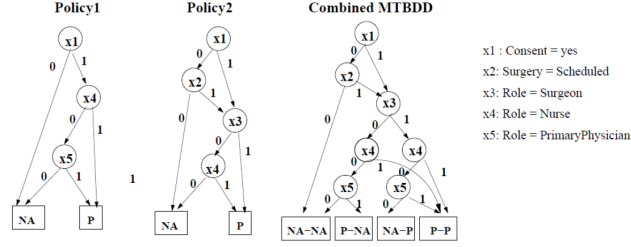


Figure 2.1: Policy MTBDD

internal nodes represent boolean predicates that correspond to: S (subject), A (action), R (resource), C (condition), and the components of a policy in which the terminals can be one of {permit, deny, not applicable} representing the effect of the policy on requests. In such diagram, the paths represent the rules of a policy. The MTBDDs of different policies are then combined to derive a single MTBDD in which each terminal corresponds to a n-tuple $\langle e_1, e_2, \dots, e_n \rangle$ that represents the effect of the policy on all requests. By browsing the paths leading to the terminals of the combined MTBDD, we can extract the set of the requests which have common or different authorizations in a given set of policies. Also, we can deduce all the requests authorized by the policies by browsing all the paths that lead to the terminal $\langle Permit, Permit \rangle$ in the combined MTBDD. For example, imagine a scenario where a patient wishes to transfer his medical records from hospital X to hospital Y each of which has its own policy, P_1 and P_2 respectively. The policy P_1 allows access to medical records if the patient has given his consent and if the subject who needs to access is either a nurse or a doctor. The policy P_2 allows access to medical records if the patient has given his consent or a surgical intervention is planned and if the subject is either a surgeon or a nurse. Before transferring the medical records, the patient must ensure that the security policy of hospital Y offers the same level of security as that of hospital X . An analysis of similarity between P_1 and P_2 can be used to ensure such a requirement. Figure 2.1 shows the MTBDDs corresponding to P_1 , P_2 and their combination. Indeed, the paths to the terminal $P - P$ in the combined MTBDD pinpoint the similarity between the two policies, whereas the remaining paths show their differences. The patient may use this information before deciding to transfer his medical records. Although this approach is precise, the computation is very intensive and it is not valid for a set of security policies (PolicySet).

Margrave [6] contains, in addition to the verification component of properties, a system for the analysis of impact change. It considers two policies and summarizes the differences between them. Margrave allows to represent the security policies in the form of trees using the same procedure as the tree that is manipulated by EXAM. The tree resulting from this combination can be further analyzed and verified, with the tool Margrave as described in [6]. Margrave defines a diagram of decision called a decision diagram of analysis change or a Common Multiple Terminal Binary Decision Diagram (CMTBDD) whose representation shows the combinations of

changes occurring between the policies as well as the differences between them. This tool has the disadvantage of not being able to model the structures of different specification languages of security policies and is limited to Ponder and XACML languages in addition to the difficulty results analysis for complex specifications.

A policy score calculation approach is proposed in [14]. The similarity measure of the proposed policy is based on the comparison of each pair of components of the corresponding policies. In fact, the measure of similarity of two policies consists in assigning a score of similarity which relies on the relationship between the sets of allowed requests (denied) by the two policies. The score of similarity is a value between 0 and 1. For example, in a scenario where a series of requests allowed (denied) by a policy P_1 is a subset of requests permitted (denied) by P_2 , the score of similarity of policies P_1 and P_2 must be higher than the score assigned to a scenario in which the whole set of the authorized requests (denied) by P_1 and P_2 have a few or no requests in common. We are going to present now how to get the score of similarity of two policies. Given two policies P_1 and P_2 , the rules of these policies are first grouped according to their effects, which results in a set of rules Permit (note PR) and a set of rules Deny (note DR). Each rule in P_1 is then compared to a rule in P_2 which has the same effect, and a score S_{rule} is then obtained which is in fact the sum of the similarity scores of the elements constituting the rules as shown in Figure 2.2. Then, based on S_{rule} , one-many mapping functions Φ for each rule in the two policies are calculated. Using Φ , the score of the entire rule is calculated. Finally, the score of overall similarity between P_1 and P_2 is obtained (see Figure 2.3). The similarity between the policies depends on the score. Although this approach cannot be used to enumerate the differences between the policies according to specific requests, it can be used as a method of quick filtering to prune the dissimilar policies before the use of analyzers of similarity which are more precise and computing-intensive. This is particularly useful when a large number of policies have to be compared.

Mazzoleni et al. [16] propose a process to calculate the similarity between two XACML policies. The process includes three stages:

- The similarity is calculated between the rules taking into account only their **Target** and **Condition**: the first step is to calculate the similarity between pairs of rules defined using the same attributes of a policy. The objective is to identify, for each attribute, the policy that specifies the most restrictive condition. To obtain this information, this method performs an analysis of the conditions of the two rules and according to the data types and functions, it extracts and compares the sets of values.
- The similarities of rules are grouped and simplified by taking into account their effects: after computing the similarities, some transformation functions are applied with a certain logic in order to reduce the number of similarities of rules. The goal of this step is to obtain only two values of similarity; one for the rules whose effect is "Deny" and the other for the rules defined with "Permit" as effect.
- The similarity of policies is extracted using the rule combining algorithms specified by

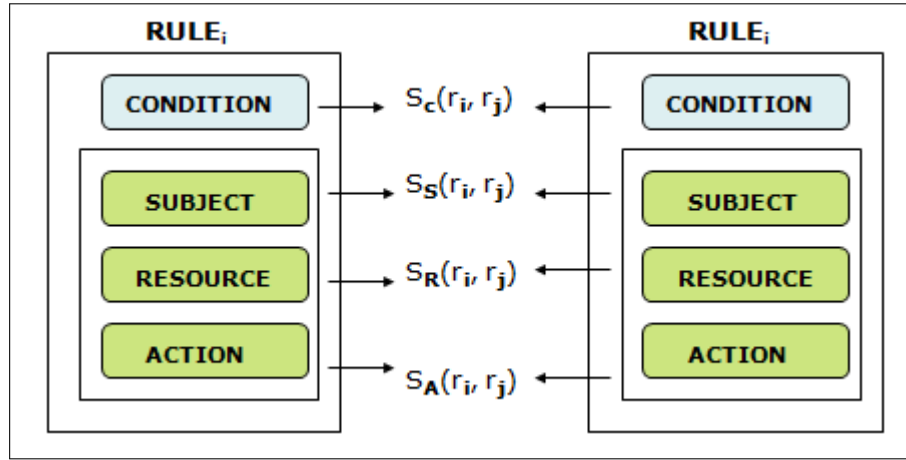


Figure 2.2: S_{rule} computation process

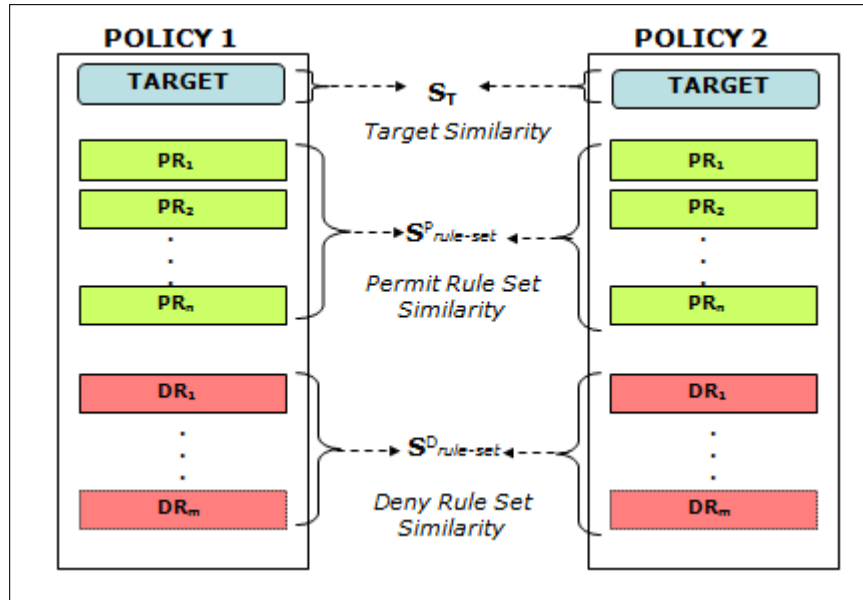


Figure 2.3: Global score computation

the two policies: let P_m and P_n be two policies and $i, j \in \{\text{Converge, Diverge, Restrict, Extend, Shuffle}\}$ be two values of similarity. If P_m specifies “Permit-overrides” and P_n specifies “Deny-overrides” as their rule combining algorithms, then the similarity between the policies P_m and P_n is calculated as follows:

- $\{R_{\text{Converge, Extend}}^{\text{Permit}}, R_i^{\text{Deny}}\} = \text{Extend},$
- $\{R_{\text{Diverge}}^{\text{Permit}}, R_i^{\text{Deny}}\} = \text{Diverge},$
- $\{R_{\text{Restrict, Shuffle}}^{\text{Permit}}, R_i^{\text{Deny}}\} = \text{Shuffle}.$

In addition, if P_m specifies “Deny-overrides” and P_n specifies “Permit-overrides” as the rule combining algorithms, then the similarity between the policies P_m and P_n is as follows:

- $\{R_{\text{Converge, Extend}}^{\text{Permit}}, R_i^{\text{Deny}}\} = \text{Restrict},$
- $\{R_{\text{Diverge}}^{\text{Permit}}, R_i^{\text{Deny}}\} = \text{Diverge},$
- $\{R_{\text{Restrict, Shuffle}}^{\text{Permit}}, R_i^{\text{Deny}}\} = \text{Shuffle}.$

This method of calculating similarity of policies is limited to policies having the same attribute. Similarly, it is not valid for a set of security policies (PolicySet) and handles only two combining algorithms: “Permit-overrides” and “Deny-overrides” of the entire set of rule combining algorithms described in XACML. In [2], Backes et al. propose an algorithm to verify the refinement of business confidentiality policies. The concept of policy refinement is close to the similarity of the policies in a certain sense, because this method checks if a policy is a subset of the other. However, their study is based on EPAL rather than XACML. It is an important difference, because the sole rule combining algorithm that is considered in their work is the First-one-applicable.

Lately, Jebbaoui et al. [10] devised a new set-based language called SBA-XACML to capture the complex structures in XACML and endowed it with a semantics to detect access flaws, conflicts and redundancies between policies. Their work covers the well-known rules combining algorithms (e.g. Permit overrides and First Applicable). However, the semantic space lacks a distance to assess the similarity between rules.

3. Description of the approach

In this section, we focus on the similarity of XACML policies. Given two policies P_1 and P_2 , the process involves two steps namely:

- Step 1: The first step is concerned with the formalization of a XACML policy as a boolean expression while taking into account the policy and rule combining algorithms. This formalization is based on a new compact and formal security language called Security Policy Language (SePL).

- Step 2: The two policy representations, which are derived from the previous step, are the input to a rewriting system to conclude which kind of relation exist between two security policies such as equivalence, restriction, inclusion, and divergence.

More precisely, we are interested in comparing security policies to know whether one is equivalent to another, greater or smaller than another, disjoint from another, etc. To this end, we proceed according to the following steps :

1. We transform every XACML property to an SePL formula.
2. We transform every SePL formula to a boolean expression. More precisely, the semantics of SePL formulae are transformed to terms in a boolean ring.
3. We generate a theorem prover from the axioms of boolean ring. This theorem prover is a confluent and convergent rewriting system that transform every tautology to the term 1.
4. We transform our question to a tautology checking query. For example, to prove whether a security policy P_1 is included in P_2 , we simply use the rewriting system to check whether the boolean ring term corresponding to $P_1 \Rightarrow P_2$ could be rewritten to 1.

The Security Policy Language (SePL) [17] is a formal and compact language for expressing security policies. All the details about the syntax and semantics of SePL are provided in the appendix.

4. XACML Security Policy Analysis

In this section, we show how we can make some formal analysis related to security properties. The steps that are aforementioned in the previous section are detailed in what follows.

4.1. From XACML to SePL

The transformation from a XACML policy into a SePL term then to an algebraic semantic term was done in a previous work that we summarize in the Appendix. Here, we give an example of two XACML policies that illustrate such transformation and which we consider in the remaining steps of the current work. The first policy P is composed of two rules. The first one states that Alice and Bob do not have the right to write in the file "secret.txt". The second one states that any action can be performed by any subject on the file "secret.txt". The two rules are combined using the first applicable algorithm. The XACML specification of P is provided in Table 1.

The second policy Q contains two rules, the first one states that Alice and Bob don't have the right to write in the file "secret.txt". The second rules states that Alice does not have the right to write in that "secret.txt" file. The two rules are combined using the First-Applicable algorithm. The XACML specification of Q is provided in Tables 2 and 3.

The formalization of the aforementioned policies in SePL is as follows.

Table 1: XACML Policy *P*

```

<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy1" Version="1.0"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:String-equal">
          <Attribute Value="secret.txt" Data Type="http://www.w3.org/2001/XMLSchema:string">
            <Attribute Designator="MustBePresent" MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-686 subject"
              AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
              Data Type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule1" Effect="Deny">
      <Target>
        <AnyOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:String-equal">
              <Attribute Value="write" Data Type="http://www.w3.org/2001/XMLSchema:string">
                <Attribute Designator="MustBePresent" MustBePresent="false"
                  Category="urn:oasis:names:tc:xacml:1.0:action-category:access-686 subject"
                  AttributeId="urn:oasis:names:tc:xacml:2.0:action:action-id"
                  Data Type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
              </Match>
            </AllOf>
          </AnyOf>
        </AnyOf>
      </Target>
      <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule2" Effect="Permit"/>
    </Rule>
  </Policy>

```

Table 2: XACML Policy *Q* (Part I)

```

<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17
http://docs.oasis-open.org/xacml/3.0/xacml-core-v3-schema-wd-17.xsd"
PolicyId="urn:oasis:names:tc:xacml:3.0:example:SimplePolicy2" Version="1.0"
RuleCombiningAlgId="urn:oasis:names:tc:xacml:1.0:rule-combining-algorithm:first-applicable">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:String-equal">
          <Attribute Value="secret.txt" Data Type="http://www.w3.org/2001/XMLSchema#string">
            <Attribute Designator="MustBePresent" MustBePresent="false"
              Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-686 subject"
              AttributeId="urn:oasis:names:tc:xacml:2.0:resource:resource-id"
              Data Type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
    <Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule1" Effect="Deny">
      <Target>
        <AnyOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:String-equal">
              <Attribute Value="write" Data Type="http://www.w3.org/2001/XMLSchema#string">
                <Attribute Designator="MustBePresent" MustBePresent="false"
                  Category="urn:oasis:names:tc:xacml:1.0:action-category:access-686 subject"
                  AttributeId="urn:oasis:names:tc:xacml:2.0:action:action-id"
                  Data Type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
              </Match>
            </AllOf>
          </AnyOf>
        </Target>
        <AnyOf>
          <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-at-least">
              <Attribute Value="Alice" Data Type="http://www.w3.org/2001/XMLSchema#string">
                <Attribute Designator="MustBePresent" MustBePresent="false"
                  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access- subject"
                  AttributeId="urn:oasis:names:tc:xacml:2.0:subject:subject-id"
                  Data Type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
              </Match>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-at-least">
              <Attribute Value="Bob" Data Type="http://www.w3.org/2001/XMLSchema#string">
                <Attribute Designator="MustBePresent" MustBePresent="false"
                  Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-686 subject"
                  AttributeId="urn:oasis:names:tc:xacml:2.0:subject:subject-id"
                  Data Type="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
              </Match>
            </AllOf>
          </AnyOf>
        </Target>
      </Rule>

```

Table 3: XACML Policy Q (Part II)

```

<Rule RuleId="urn:oasis:names:tc:xacml:3.0:example:SimpleRule2" Effect="Permit">
  <Target>
    <AnyOf>
      <AllOf>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:String-equal">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            write</AttributeValue> <AttributeDesignator MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:action-category:access-686 subject"
            AttributeId="urn:oasis:names:tc:xacml:2.0:action:action-id"
            DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </Match>
        <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-at-least">
          <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">
            Alice</AttributeValue> <AttributeDesignator MustBePresent="false"
            Category="urn:oasis:names:tc:xacml:1.0:subject-category:access- subject"
            AttributeId="urn:oasis:names:tc:xacml:2.0:subject:subject-id"
            DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </Match>
        </AllOf>
      </AnyOf>
    </Target>
  </Rule>
</Policy>

```

- For the first policy P , we have $P = \mathbf{FA}(\phi : (P_1, P_2))$
where

$$\begin{aligned}
 \phi &= (x) \\
 P_1 &= (y, z) \rightarrow \mathbf{d} \\
 P_2 &= () \rightarrow \mathbf{p} \\
 x &= \mathbf{string-equal}(AttResource, secret.txt) \\
 y &= \mathbf{string-equal}(AttAction, write) \\
 z &= \mathbf{string-at-least}(AttSubject, Alice) \text{ or } \\
 &\quad \mathbf{string-at-least}(AttSubject, Bob)
 \end{aligned}$$

- From abbreviation 2 (Section 7.3), it follows:

$$\begin{aligned}
 P_1 &= (\perp, (y, z)) \\
 P_2 &= ((), \perp)
 \end{aligned}$$

- By abbreviation 2 (Section 7.3), it follows:

$$P = \mathbf{FA}(\phi : (P_1, P_2)) = \mathbf{FA}((\perp, (x, y, z)), ((x), \perp))$$

- From the definition of \mathbf{FA} , it follows:

$$P = ((\perp, (x, y, z)).((x), \perp))$$

- For the second policy Q , we have: $Q = \mathbf{FA}(\phi : (Q_1, Q_2))$
where

$$\begin{aligned}
\phi &= (x) \\
Q_1 &= (y, z) \rightarrow \mathbf{d} \\
Q_2 &= (y, u) \rightarrow \mathbf{p} \\
x &= \mathbf{string-equal}(AttResource, secret.txt) \\
y &= \mathbf{string-equal}(AttAction, write) \\
z &= \mathbf{string-at-least}(AttSubject, Alice) \text{ or } \mathbf{string-at-least}(AttSubject, Bob) \\
u &= \mathbf{string-at-least}(AttSubject, Alice)
\end{aligned}$$

As we did for P , using the shortcuts of Section 7.3, it follows that:

$$Q = ((\perp, (x, y, z)).((x, y, u), \perp))$$

4.2. From SePL to Boolean Ring

Based on the semantics of SePL, we transform a policy in SePL to a tuple of terms in the boolean ring. For our purpose, the boolean-ring formalism is more interesting than the boolean algebra since its axioms define a unique normal form (up to associativity and commutativity of the two operators) which is a key feature for building a theorem prover. However, there is an intrinsic link between boolean algebra and boolean ring as shown hereafter.

- From each boolean ring, we can construct a boolean algebra:
If $(B, \oplus, *, 0, 1)$ is a boolean ring, then $(B, \vee, \wedge, \neg, 0, 1)$ is a boolean algebra such that \vee, \wedge, \neg are defined as shown in Table 4.

$$\begin{aligned}
x \vee y &= x \oplus y \oplus x * y \\
x \wedge y &= x * y \\
\neg x &= x \oplus 1
\end{aligned}$$

Table 4: From Rings to Algebra

- Inversely, from a boolean algebra we can construct a boolean ring:
If $(B, \vee, \wedge, \neg, 0, 1)$ is a boolean algebra, then $(B, \oplus, *, 0, 1)$ is a boolean ring such that \oplus and $*$ are defined in Table 5.

$$\begin{aligned}
x \oplus y &= (x \wedge \neg y) \vee (\neg x \wedge y) \\
x * y &= x \wedge y
\end{aligned}$$

Table 5: From Algebra to Rings

The semantics of SePL transforms a security policy to a tuple of terms in the boolean algebra. We use the link between boolean ring and boolean algebra to transform the semantic term into a term in boolean Ring. If we have many policies, we should make sure that variables involved in terms are completely disjoint. For instance, if the variable x means $subject - id = \{Alice\}$ and y means $subject - id = \{Alice, Bob\}$, then we should create a new variable z that is equal to $subject - id = \{Bob\}$ and substitute y by $x \vee z$ to obtain disjoint variables.

Example. Let us continue with our previous example. We already transformed P and Q for XACML to SePL and we obtained:

- $P = ((\perp, (x, y, z)).((x), \perp))$
- $Q = ((\perp, (x, y, z)).((x, y, u), \perp))$

where

$$\begin{aligned} x &= \text{string-equal}(AttResource, secret.txt) \\ y &= \text{string-equal}(AttAction, write) \\ z &= \text{string-at-least}(AttSubject, Alice) \text{ or } \\ &\quad \text{string-at-least}(AttSubject, Bob) \\ u &= \text{string-at-least}(AttSubject, Alice) \end{aligned}$$

We observe that the variables z and u are not disjoint. To make them disjoint, we create a new variable $w = \text{string-at-least}(AttSubject, Bob)$ and we substitute z by $u \vee w$. Then, we obtain:

$$\begin{aligned} P &= ((\perp, (x, y, u \text{ or } w)).((x), \perp)) \\ Q &= ((\perp, (x, y, u \text{ or } w)).((x, y, u), \perp)) \end{aligned}$$

Using the definition of semantics, we obtain:

$$\begin{aligned} \llbracket P \rrbracket &= (F_3 \vee ((x, T, T) - (x, y, u \vee w)), (x, y, u \vee w) \vee (F_3 - F_3)) \\ \llbracket Q \rrbracket &= (F_3 \vee ((x, y, u) - (x, y, u \vee w)), (x, y, u \vee w) \vee (F_3 - F_3)) \end{aligned}$$

where $F_n, n \in \mathbb{N}^+$, is an abbreviation of $\underbrace{(F, \dots, F)}_n$, (e.g., $F_3 = (F, F, F)$).

Now, we have tuples of boolean terms that we can use as input to the theorem prover.

4.3. Theorem Prover

The theorem prover is a confluent and terminating rewrite system build mainly from the axioms of a boolean ring. Contrarily to a boolean algebra, a boolean ring defines a unique normal form, up to associativity and commutativity of the two operators, for every term. Tautologies reduce all the the time to 1, contradictions to 0, and formulae that can be satisfied under some conditions are reduced to neither.

Table 6: Main Inference rules

F	\rightarrow	0	R_0
T	\rightarrow	1	R_1
$x - y$	\rightarrow	$x \oplus x * y$	R_2
$x \vee y$	\rightarrow	$x * y \oplus x \oplus y$	R_3
$x \wedge y$	\rightarrow	$x * y$	R_4
$x \implies y$	\rightarrow	$x * y \oplus x \oplus 1$	R_5
$x \approx y$	\rightarrow	$x \oplus y \oplus 1$	R_6
$\neg x$	\rightarrow	$x \oplus 1$	R_7
$x \oplus 0$	\rightarrow	x	R_8
$x \oplus x$	\rightarrow	0	R_9
$x * 1$	\rightarrow	x	R_{10}
$x * x$	\rightarrow	x	R_{11}
$x * 0$	\rightarrow	0	R_{12}
$x * (y \oplus z)$	\rightarrow	$x * y \oplus x * z$	R_{13}

The main rules of our theorem prover are given in Table 6. Rules R_0, \dots, R_7 transform boolean expressions returned by our semantics to terms in the boolean ring. Rules R_8, \dots, R_{13} are axioms of the boolean ring.

We extend the aforementioned rewriting rules to deal with a n-tuple by new ones given by Table 7.

Table 7: Rules Extension

F_n	\rightarrow	(F, \dots, F)	T_0
T_n	\rightarrow	(T, \dots, T)	T_1
$(x_1, \dots, x_n) - (y_1, \dots, y_n)$	\rightarrow	$(x_1 - y_1, \dots, x_n - y_n)$	T_2
$(x_1, \dots, x_n) \vee (y_1, \dots, y_n)$	\rightarrow	$(x_1 \vee y_1, \dots, x_n \vee y_n)$	T_3
$(x_1, \dots, x_n) \wedge (y_1, \dots, y_n)$	\rightarrow	$(x_1 \wedge y_1, \dots, x_n \wedge y_n)$	T_4
$(x_1, \dots, x_n) \implies (y_1, \dots, y_n)$	\rightarrow	$(x_1 \implies y_1, \dots, x_n \implies y_n)$	T_5
$(x_1, \dots, x_n) \approx (y_1, \dots, y_n)$	\rightarrow	$(x_1 \approx y_1, \dots, x_n \approx y_n)$	T_6
$\neg(x_1, \dots, x_n)$	\rightarrow	$(\neg x_1, \dots, \neg x_n)$	T_7
$(x_1, \dots, 0, \dots, x_n)$	\rightarrow	0	T_8
$(1, \dots, 1)$	\rightarrow	1	T_9

The previous rules should be applied under commutativity and associativity of $*$ and \oplus , i.e:

$$\begin{array}{lll}
\text{Commutativity} & x \oplus y & = y \oplus x \\
& x * y & = y * x \\
\text{Associativity} & (x \oplus y) \oplus z & = x \oplus (y \oplus z) \\
& (x * y) * z & = x * (y * z)
\end{array}$$

4.4. Queries and proofs

Our query should be transformed to a tautology test in a boolean ring or a boolean algebra. Since, we are interested into comparing security policies, we can easily transform different kind of queries to tautology tests as shown in Table 8 .

Example. Let us continue with our previous example. We have :

$$\begin{aligned}
\llbracket P \rrbracket &= (F_3 \vee ((x, T, T) - (x, y, u \vee w)), (x, y, u \vee w) \vee (F_3 - F_3)) \\
\llbracket Q \rrbracket &= (F_3 \vee ((x, y, u) - (x, y, u \vee w)), (x, y, u \vee w) \vee (F_3 - F_3))
\end{aligned}$$

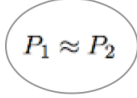
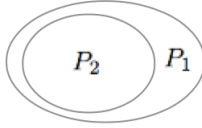
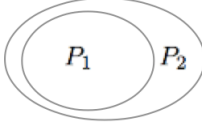
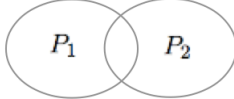
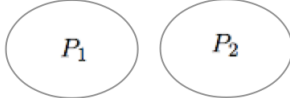
Suppose that we want to check whether P is equivalent to Q or not. Our query is $P \approx Q$.

$$P \approx Q \text{ if } \left\{ \begin{array}{ll} F_3 \vee ((x, T, T) - (x, y, u \vee w)) & \approx F_3 \vee ((x, y, u) - (x, y, u \vee w)) \\ & \text{and} \\ (x, y, u \vee w) \vee (F_3 - F_3) & \approx (x, y, u \vee w) \vee (F_3 - F_3) \end{array} \right.$$

The second equivalence is trivial. Let us prove the first one using the inference system:

$$\begin{aligned}
& F_3 \vee ((x, T, T) - (x, y, u \vee w)) \approx F_3 \vee ((x, y, u) - (x, y, u \vee w)) \\
& \text{By Applying } T_0 \text{ and } R_0, \text{ we get} \\
& \rightarrow (0, 0, 0) \vee ((x, T, T) - (x, y, u \vee w)) \approx (0, 0, 0) \vee ((x, y, u) - (x, y, u \vee w)) \\
& \text{By Applying } T_1 \\
& \rightarrow (0, 0, 0) \vee ((x, 1, 1) - (x, y, u \vee w)) \approx (0, 0, 0) \vee ((x, y, u) - (x, y, u \vee w)) \\
& \text{By Applying } T_2 \\
& \rightarrow (0, 0, 0) \vee (x - x, 1 - y, 1 - (u \vee w)) \approx (0, 0, 0) \vee (x - x, y - y, u - (u \vee w)) \\
& \text{By Applying } R_2 \\
& \rightarrow (0, 0, 0) \vee (x \oplus x * x, 1 - y, 1 - (u \vee w)) \approx (0, 0, 0) \vee (x \oplus x * x, y - y, u - (u \vee w)) \\
& \text{By Applying } R_{11} \\
& \rightarrow (0, 0, 0) \vee (x \oplus x, 1 - y, 1 - (u \vee w)) \approx (0, 0, 0) \vee (x \oplus x, y - y, u - (u \vee w)) \\
& \text{By Applying } R_9 \\
& \rightarrow (0, 0, 0) \vee (0, 1 - y, 1 - (u \vee w)) \approx (0, 0, 0) \vee (0, y - y, u - (u \vee w)) \\
& \text{Appliquons } T_8 \\
& \rightarrow (0, 0, 0) \vee (0, 0, 0) \approx (0, 0, 0) \vee (0, 0, 0) \\
& \text{By Applying } R_3 \\
& \rightarrow (0 * 0 \oplus 0 \oplus 0, 0 * 0 \oplus 0 \oplus 0, 0 * 0 \oplus 0 \oplus 0) \approx (0 * 0 \oplus 0 \oplus 0, 0 * 0 \oplus 0 \oplus 0, 0 * 0 \oplus 0 \oplus 0) \\
& \text{By Applying } R_{12}
\end{aligned}$$

Table 8: Comparing Security Policies

Case	Figure	Term to prove
P_1 converges P_2		$P_1 \approx P_2$
P_1 extends P_2		$(P_1 \Rightarrow P_2) \wedge \neg(P_2 \Rightarrow P_1)$
P_1 restricts P_2		$(P_2 \Rightarrow P_1) \wedge \neg(P_1 \Rightarrow P_2)$
P_1 shuffles P_2		$\neg(P_1 \oplus P_2) \wedge \neg(P_2 \Rightarrow P_1) \wedge \neg(P_1 \Rightarrow P_2)$
P_1 diverges P_2		$P_1 \oplus P_2$

$\rightarrow (0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0) \approx (0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0, 0 \oplus 0 \oplus 0)$

By Applying R_8

$\rightarrow (0, 0, 0) \approx (0, 0, 0)$

By Applying T_6

$\rightarrow (0 \approx 0, 0 \approx 0, 0 \approx 0)$
 By Applying R_6
 $\rightarrow (0 \oplus 0 \oplus 1, 0 \oplus 0 \oplus 1, 0 \oplus 0 \oplus 1)$
 By Applying R_8
 $\rightarrow (1, 1, 1)$
 By Applying T_9
 $\rightarrow 1$

Finally, we conclude that the two policies are equivalent.

5. Policies Similarity Algorithm

We present in this section our policies similarity algorithm (Algorithm 1). The algorithm takes as input two XACML policies, transforms them into SePL terms based on the function $\lceil - \rceil$ that is presented in Table 7.4. The semantics function $\llbracket \cdot \rrbracket$ transforms the SePL terms into terms of a boolean algebra, which are mapped into terms of a boolean ring using the transformation described in Section 5. These terms are the input to the theorem prover to deduce the similarity type (Converge, Diverge, Extend, Shuffle) between the two policies.

Algorithm 1 Policies Similarity Algorithm

Require: XACML policies XP_1 and XP_2

Ensure: Policy similarity type

$SP_i = \lceil XP_i \rceil \quad \triangleright \lceil - \rceil$ is the transformation from a XACML policy to a SePL term as presented in Table 7.4

$SP_i = \llbracket P_i \rrbracket \quad \triangleright \llbracket P_i \rrbracket$ denotes the semantics of the policy P_i , which is a term in a boolean algebra

$BP_i = \mathcal{B}(SP_i) \quad \triangleright \mathcal{B}$ denotes the transformation of a term in an algebra to a boolean ring as early explained in Table 5

SimType = "Converge" if $BP_1 \approx BP_2$

SimType = "Diverge" if $BP_1 \oplus BP_2$

SimType = "Restrict" if $BP_2 \implies BP_1 \wedge \neg(BP_1 \implies BP_2)$

SimType = "Extend" if $BP_1 \implies BP_2 \wedge \neg(BP_2 \implies BP_1)$

SimType = "Shuffle" if $BP_1 \oplus BP_2 \wedge \neg(BP_1 \implies BP_2) \wedge \neg(BP_2 \implies BP_1)$

return SimType

The worst case time complexity of the transformation algorithms is $O(m)$, where m is the number of rules. The complexity of the policies similarity algorithm is $O(n \times m)$, where n is the number of policies. So our complexity is not exponential, which shows that our algorithm has a reasonable complexity.

6. Experimental Evaluation

To analyze the performance of the policies similarity process, we developed a prototype using Java and XACML. More precisely, we used Tom; a pre-compiler that enables us to implement our rewriting system and transformation rules. Figure 6.1 shows the interface of policies similarity that we have implemented as part of the whole tool.

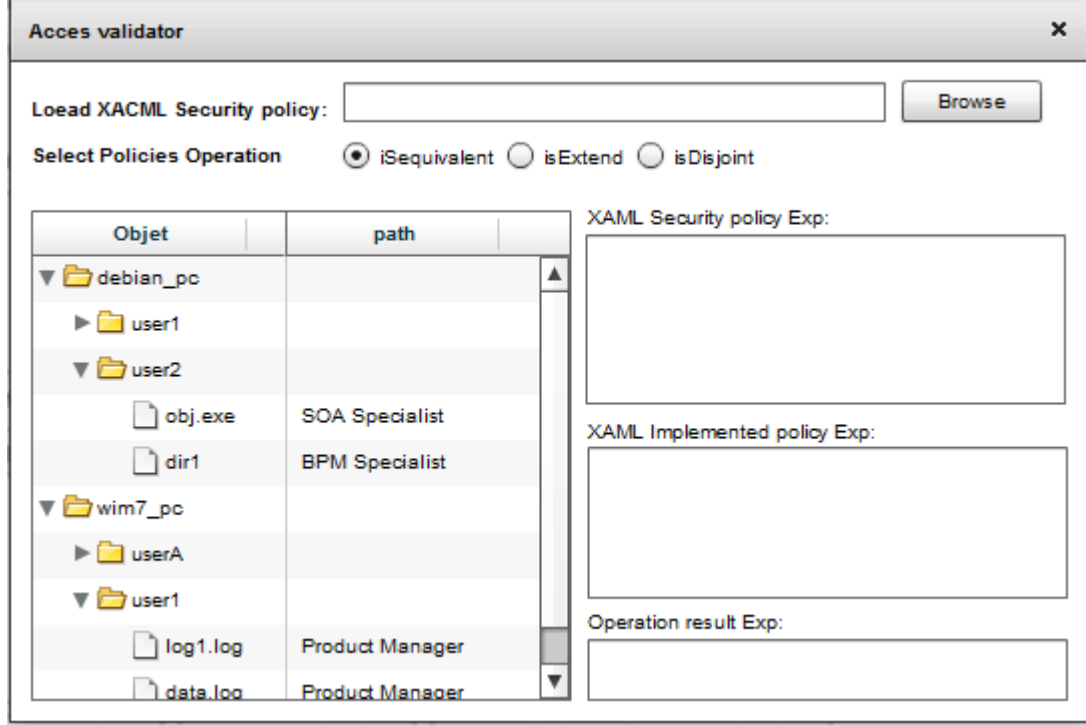


Figure 6.1: Policy similarity interface

We present evaluate the performance of our policies similarity algorithm.

6.1. Performance analysis

To evaluate our approach, we developed a tool that generates XACML policies with a random number of rules. The experiments presented in this section are performed 10 times on a machine Core i5, 2.30 Ghz, 4 Go of RAM with a Windows 7 OS. The average of the obtained results is then computed.

We first do the experiments for the evaluation of our rewriting system, which translates a XACML policy into a boolean expression. The figure below shows the performance of the system for each type of similarity. The trends of the five curves are almost the same. Furthermore, the curves are linear with respect to the number of rules in a policy.

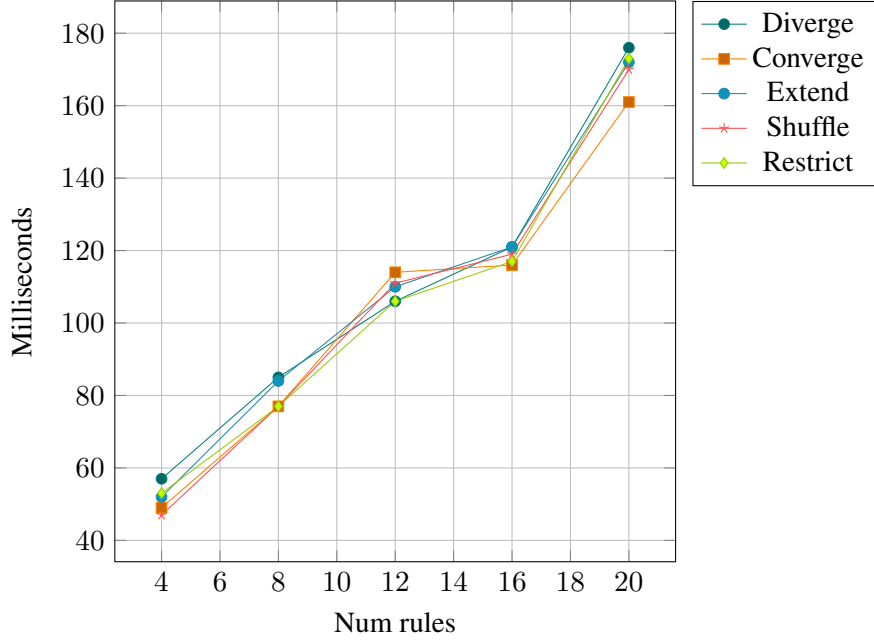


Figure 6.2: CPU Execution average time for each type of similarity

6.1.1. Preprocessing time

We evaluate here the time required to rewrite a policy as a term in a boolean ring for a pair of policies. The average number of parameters (resource, action et subject) varies between 20 et 100 for each policy and the number of rules varies between 8 and 32. The preprocessing time is linear with respect to the number of rules and parameters. More precisely, if each policy has 32 rules and 100 parameters, the preprocessing time is 90% the global response time. Hence, the global similarity response time is dominated by the preprocessing module.

6.1.2. Response time

We evaluate now the total time for policies similarity. The figure below shows the time variation while increasing the number of compared policies.

Knowing that the number of pairs is generally between 20 and 100 [15], our approach has a reasonable response time. Another experiment is done by fixing the number of elements in each policy and varying the number of policy pairs to be compared. The policies have 10 parameters in average. We did the experiment for 8 to 16 rules.

We observe that the time required to compare few hundred of policies is few seconds. The minimal response time is 1s and the maximal is 1m. This shows the feasibility and scalability of our approach.

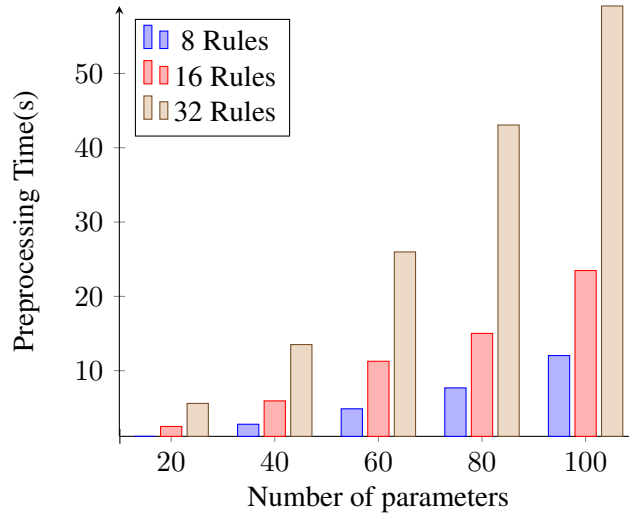


Figure 6.3: Preprocessing time for a pair of policies

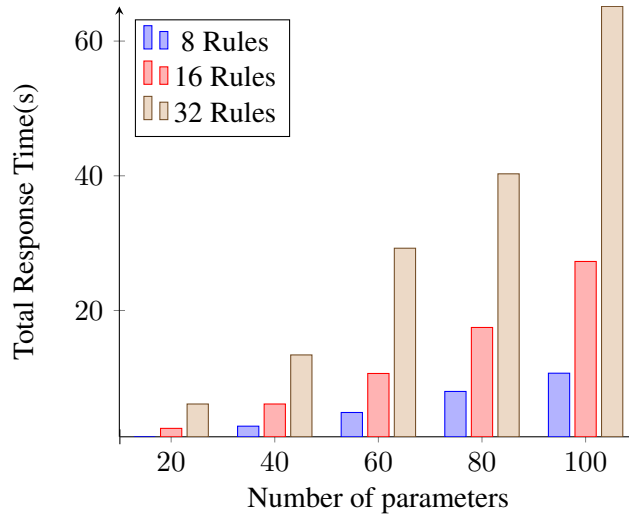


Figure 6.4: Total response time to compare a pair of policies

7. Conclusion and future work

We presented a new approach for the assessment of policies similarity that is based on a rewriting system. Each policy is formalized as SePL expression then mapped into a term in a boolean ring based on pre-defined transformation rules. We presented a case study, which shows

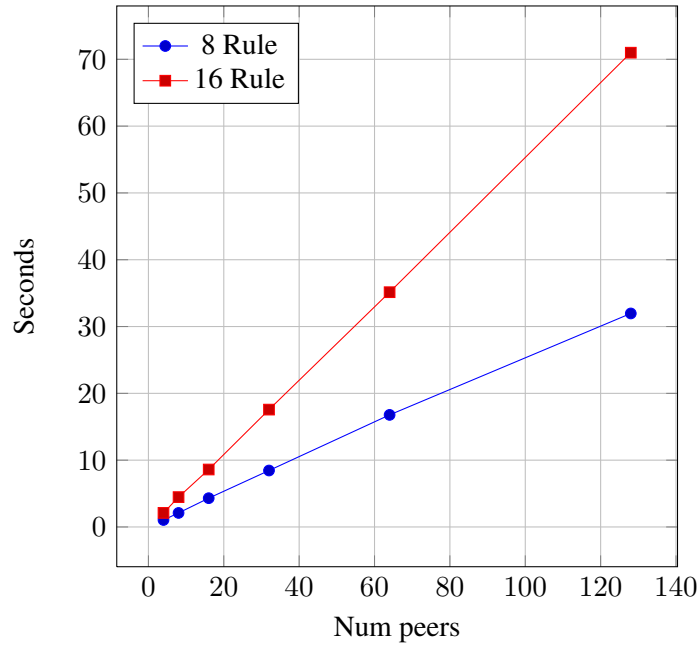


Figure 6.5: CPU execution average time

how to compare two XACML policies based on our approach. We provided also a complexity and performance analysis of our global algorithm. Our future work will be oriented towards the optimization of the proposed approach and its extension to deal with more rule combining algorithms.

References

References

- [1] B, L. and W. (1971). Protection. *In proc. 5th Princeton Conf.on information Sciences and systems*.
- [2] Backes, M., Karjoth, G., Bagga, W., and Schunter, M. (2004). Efficient comparison of enterprise privacy policies. *In Proceedings of the 2004 ACM Symposium on Applied Computing (SAC)*, pages 375–382.
- [3] Bell,E and Lapadula (1976). secure computer systems : Unified exposition and multics interpretation. *the MITRE Corporation, Technical report*.
- [4] Cook, S. A. (1971). The complexity of theorem-proving procedures. *3rd. ACM Symp. on Theory of Computing*, pages 151–158.

- [5] Dershowitz, N., Hsiang, J., shieng Huang, G., and Kaiss, D. (2004). Boolean ring satisfiability. In *in International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*.
- [6] Fisler, K., Krishnamurthi, S., Meyerovich, L. A., and Tschantz, M. C. (2005). Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering (ICSE)*, pages 196–205.
- [7] Harrison, Ullman, and Ruzzo (1976). Protecting in operating system. *Communication ACM*.
- [8] Hsiang, J. and Dershowitz, N. (1983). Rewrite methods for clausal and nonclausal theorem proving. In *in Proceedings of the Tenth International Conference on Automata, Languages and Programming*. Springer Verlag.
- [9] Jajodia, S., Samarati, P., Sapino, M. L., and Subrahmanian, V. S. (June 2001). *Flexible support for multiple access control policies*. ACM Transactions on Database Systems.
- [10] Jebbaoui, H., Mourad, A., Otrok, H., and Haraty, R. (2015). Semantics-based approach for detecting flaws, conflicts and redundancies in xacml policies. *Computers & Electrical Engineering*, 44:91–103.
- [11] Jordan, C. (1987). A guide to understanding discretionary access control in trusted systems. Technical report, National Computer Security Center.
- [12] K.J, B. (1977). Integrity consideration for secure computer systems, the mitre corporation. *Technical Report ESD-TR-76-372, MTR-3153*.
- [13] Koch, M., Mancini, L. V., and P.-Presicce, F. (2001). On the specification and evolution of access control policies. In *Proceedings of the 6th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 121–130.
- [14] Lin, D., Rao, P., Bertino, E., and Lobo, J. (2007). An approach to evaluate policy similarity. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 1–10.
- [15] Lindan, D., Rao, P., Bertino, E., ninghui, and Lobo, J. (2007). Exam a comprehensive environment for the analysis of access control policies. In *Technical Report*.
- [16] Mazzoleni, P., Bertino, E., and Crispo, B. (2006). Xacml policy integration algorithms. In *Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT)*, pages 223–232.
- [17] Mejri, M. and Yahyaoui, H. (2016). Formal specification and integration of distributed security policies. <http://arxiv.org/abs/1605.06233>.

- [18] Samarati, P. and de Capitani di Vimercati, S. (2001). Access control: Policies, models, and mechanisms. *FOSAD 2000, LNCS 2171*, pages 137–196.
- [19] Sandhu, R. S., Coyne, E. J., Feinstein, H. L., and Youman, C. E. (1996). Role-based access control models. *IEEE Computer*, pages 38–47.
- [20] Specification, C. (2010). extensible access control markup language (xacml) version 3.0. *OASIS*.

Appendix

7.1. Syntax

We present in this Section a new security policy language (SePL) that is compact and expressive. This language is generic and is meant to be used to perform security policy integration. The syntax of SePL is given by the following BNF grammar:

$$\begin{aligned}
 P, P_1, P_2 &::= \varepsilon \mid 0 \mid 1 \mid R \mid \neg P \mid \lceil P \rceil \mid P_1 \cdot P_2 \\
 &\quad \mid P_1 \parallel P_2 \mid P_1 \ll P_2 \mid P_1 \parallel P_2 \\
 &\quad \mid P_1 + P_2 \mid P_1 - P_2 \mid P_1 \ominus P_2 \\
 R &::= \langle \varphi_1, \varphi_2 \rangle
 \end{aligned}$$

where

- ε denotes the empty policy.
- 0 denotes the policy that denies all actions.
- 1 denotes the policy that accepts all actions.
- $R = \langle \varphi_1, \varphi_2 \rangle$ denotes the policy that accepts actions accepted by φ_1 and not denied by φ_2 and denies actions denied by φ_2 and not accepted by φ_1 .
- $\neg P$ denotes the policy that accepts actions that P denies and denies actions that P accepts.
- $\lceil P \rceil$ behaves like P except that it transforms the indeterminate part of P to not applicable. In other words, if the accept part or the deny part of P is indeterminate, it becomes an empty set.
- $P_1 \cdot P_2$ denotes the sequential composition of policies. This gives the result of the first one applicable. If no one is applicable, the result is either not applicable or indeterminate if P_1 and P_2 are indeterminate.

- $P_1 \parallel P_2$ denotes the policy that gives priority to accept, i.e, it accepts an action when P_1 or P_2 accepts it and denies an action when no one of them accepts it and at least one of them denies it. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.
- $P_1 \parallel\!\!\!\! \parallel P_2$: This is the dual part of \parallel since it gives priority to deny. It denies an action when P_1 or P_2 denies it and accepts an action when no one of them denies it and at least one of them accepts it. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.
- $P_1 \parallel P_2$ denotes the parallel composition of policies. It accepts an action when both of them accept and denies when both of them deny. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.
- $P_1 + P_2$ denotes the choice between two policies. It accepts when one of them accept and no one denies, and denies when one of them denies and no one accepts. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.
- $P_1 - P_2$ denotes the policy that accepts when P_1 accepts and P_2 is not applicable and denies when P_1 denies and P_2 is not applicable. Otherwise, the policy is either not applicable or indeterminate if P_1 or P_2 is indeterminate.
- $P_1 \ominus P_2$. This behaves like $P_1 - P_2$ except that the result is indeterminate ("?)") when there is an overlap between P_1 and P_2 (i.e. the accepts of P_1 together with its denies is not disjoint from the accepts of P_2 added to its denies) .

To reduce the number of parentheses when writing properties, we assume the following precedence between operators (from strong to weak) $\neg, \lceil, \cdot, \parallel, \parallel\!\!\!\! \parallel, +$. For example $P_1 + \neg P_2.P_3 \parallel P_4$ is same as $P_1 + (((\neg P_2).P_3) \parallel P_4)$. Notice that SePL does not explicitly contain the composition rules (combining algorithms) used in XACML, but, as we show later that, it is expressive enough to specify them. Also, the operators of SePL are not independent from each others. We show later that we can remove many of them without affecting the expressiveness of the language. In fact, we can keep only the operators belonging to the set $\{\neg, \lceil, \parallel, +, \ominus\}$ without affecting the expressiveness of the language.

7.2. Semantics

- **Preliminary definitions:** Let $A = (\alpha_1, \dots, \alpha_n)$ and $B = (\beta_1, \dots, \beta_n)$ be two tuples of n set elements. In the sequel, we define the following semantic operators:

$$\begin{aligned}
A \cup B &= (\alpha_1 \cup \beta_1, \dots, \alpha_n \cup \beta_n) \\
A \cap B &= (\alpha_1 \cap \beta_1, \dots, \alpha_n \cap \beta_n) \\
A - B &= (\alpha_1 - \beta_1, \dots, \alpha_n - \beta_n) \\
A \ominus B &= (\alpha_1 \ominus \beta_1, \dots, \alpha_n \ominus \beta_n) \\
\bar{A} &= (\alpha_1^\neg, \dots, \alpha_n^\neg) \\
\lceil A &= (\lceil \alpha_1, \dots, \lceil \alpha_n)
\end{aligned}$$

where

$$\begin{aligned}
\alpha \ominus \beta &= \alpha - ((\alpha \cap \beta) \cap ?) \\
&= \alpha \cap \overline{((\alpha \cap \beta) \cap ?)} \\
&= \begin{cases} \alpha & \text{if } \alpha \cap \beta = \emptyset \\ ? & \text{otherwise} \end{cases}
\end{aligned}$$

$$\lceil \alpha = \begin{cases} \alpha & \text{if } \alpha \neq ? \\ \emptyset & \text{otherwise} \end{cases}$$

$$\alpha^\neg = \begin{cases} \alpha & \text{if } \alpha \neq ? \\ \mathcal{D}(\alpha) & \text{otherwise} \end{cases}$$

$$\begin{aligned}
\lceil (\alpha \cup \beta) &= \lceil \alpha \cup \lceil \beta \\
\lceil (\alpha \cap \beta) &= \lceil \alpha \cap \lceil \beta \\
\lceil (\bar{\alpha}) &= \overline{\lceil \alpha} \\
(\alpha \cup \beta)^\neg &= \alpha^\neg \cup \beta^\neg \\
(\alpha \cap \beta)^\neg &= \alpha^\neg \cap \beta^\neg \\
(\bar{\alpha})^\neg &= \overline{\lceil \alpha}
\end{aligned}$$

- **Absolute Semantics** ($\llbracket - \rrbracket$): The absolute semantics of a policy P , denoted by $\llbracket P \rrbracket$ returns a pair (A, B) where A is the acceptance domain of P and B is its denying domain. The domain that is not explicitly accepted or denied by a policy defines implicitly its “non-applicable” domain. More formally, $\llbracket - \rrbracket$ is inductively defined as shown by Table 9 where (A_1, D_1) denotes the semantics of P_1 , (A_2, D_2) denotes the semantics of P_2 and (A, D) denotes the semantics of P .

The absolute semantics gives the meaning of a policy independent from the environment in which it will be evaluated. One of the advantages of this semantics is that any optimization or simplification applied to it can be used for any environment. This means that we can reduce the time of the evaluation of the semantics by optimizing this semantics one time and use it many times with different environments. Another advantage is that it allows to prove some general results independent from a specific environment.

- **Relative Semantics** ($\llbracket - \rrbracket_\Gamma$): The semantics of a policy P relative to an environment Γ , denoted by $\llbracket P \rrbracket_\Gamma$, returns the decision of the policy P regarding Γ . In practice, the

Table 9: Absolute Semantics

$$\begin{aligned}
\llbracket < \varphi_1, \varphi_2 > \rrbracket &= (\varphi_1 - \varphi_2, \varphi_2 - \varphi_1) \\
\llbracket \varepsilon \rrbracket &= (\perp, \perp) \\
\llbracket 0 \rrbracket &= (\perp, \top) \\
\llbracket 1 \rrbracket &= (\top, \perp) \\
\llbracket \neg P \rrbracket &= (D, A) \\
\llbracket \ulcorner P \urcorner \rrbracket &= (\ulcorner A \urcorner, \ulcorner D \urcorner) \\
\llbracket P_1 \cdot P_2 \rrbracket &= (A_1 \cup (A_2 - D_1), D_1 \cup (D_2 - A_1)) \\
\llbracket P_1 \llbracket P_2 \rrbracket \rrbracket &= (A_1 \cup A_2, (D_1 - A_2) \cup (D_2 - A_1)) \\
\llbracket P_1 \rrbracket P_2 \rrbracket &= (A_1 - D_2 \cup A_2 - D_1, D_1 \cup D_2) \\
\llbracket P_1 \parallel P_2 \rrbracket &= (A_1 \cap A_2, D_1 \cap D_2) \\
\llbracket P_1 + P_2 \rrbracket &= ((A_1 \cup A_2) - (D_1 \cup D_2), (D_1 \cup D_2) - (A_1 \cup A_2)) \\
\llbracket P_1 - P_2 \rrbracket &= (A_1 - (A_2 \cup D_2), D_1 - (A_2 \cup D_2)) \\
\llbracket P_1 \ominus P_2 \rrbracket &= (A_1 \ominus (A_2 \cup D_2), D_1 \ominus (A_2 \cup D_2))
\end{aligned}$$

environment Γ contains both the action that we want to execute together with its context (the values of the variables of the environment during its execution). For our semantics, the result is (α, β) where $\{\alpha, \beta\} \subset \{\text{T}, ?, \text{F}\}$ (T stands for **True**, F stands for **False**, and $?$ stands for unknown). Usually, the result is considered as permit if $\alpha = \text{T}$, deny if $\beta = \text{T}$, otherwise the result is either not applicable or indeterminate. It is non applicable if $\alpha = \beta = \text{F}$ and it is indeterminate if α and β are in $\{(\text{F}, ?), (?, \text{F}), (?, ?)\}$. More details about these notations will be given within the section regarding the formalization of the XACML language.

More formally, let $\Gamma = [a_1 = v_1, \dots, a_n = v_n]$ be an environment where $v_i \in \mathcal{D}(a_i) \cup \{?\}$ and “ $a = ?$ ” means that the value of the attribute a is unknown. Let P be a policy such that $\llbracket P \rrbracket = (A, D)$. We extend the definition of $\llbracket - \rrbracket$, as follows:

$$\llbracket P \rrbracket_{\Gamma} = (\llbracket A \rrbracket_{\Gamma}, \llbracket D \rrbracket_{\Gamma})$$

$$\llbracket (d_1, \dots, d_n) \rrbracket_{\Gamma} = \llbracket d_1 \rrbracket_{\Gamma(a_1)} \wedge \dots \wedge \llbracket d_n \rrbracket_{\Gamma(a_n)}$$

$$\llbracket d \rrbracket_{\Gamma(a)} = \begin{cases} \text{T} & \text{if } \Gamma(a) \in d \text{ or } d = \mathcal{D}(a) \\ \text{F} & \text{if } \Gamma(a) \notin d \text{ or } d = \emptyset \\ ? & \text{otherwise} \end{cases}$$

Notice that the truth tables of the three-valued logic is as shown hereafter:

\wedge	T	?	F
T	T	?	F
?	?	?	F
F	F	F	F

\vee	T	?	F
T	T	T	T
?	T	?	?
F	T	?	F

\ominus	T	?	F
T	?	?	T
?	?	?	?
F	F	F	F

b	$\neg b$
T	F
?	?
F	T

b	\bar{b}
T	T
?	F
F	F

It is sometimes useful to define $\llbracket - \rrbracket_{\Gamma}$ in a compositional way as shown in Table 10 where (a_1, d_1) denotes $\llbracket P_1 \rrbracket_{\Gamma}$, (a_2, d_2) denotes $\llbracket P_2 \rrbracket_{\Gamma}$ and (a, d) denotes $\llbracket P \rrbracket_{\Gamma}$:

where $a - b$ denotes $a \wedge \neg b$

Table 10: Relative Semantics

$\llbracket < \varphi_1, \varphi_2 > \rrbracket_\Gamma$	$=$	$(\llbracket \varphi_1 - \varphi_2 \rrbracket_\Gamma, \llbracket \varphi_2 - \varphi_1 \rrbracket_\Gamma)$
$\llbracket \varepsilon \rrbracket$	$=$	(\mathbf{F}, \mathbf{F})
$\llbracket 0 \rrbracket_\Gamma$	$=$	(\mathbf{F}, \mathbf{T})
$\llbracket 1 \rrbracket_\Gamma$	$=$	(\mathbf{T}, \mathbf{F})
$\llbracket \neg P \rrbracket_\Gamma$	$=$	(d, a)
$\llbracket \bar{P} \rrbracket_\Gamma$	$=$	(\bar{a}, \bar{d})
$\llbracket P_1 \cdot P_2 \rrbracket_\Gamma$	$=$	$(a_1 \vee (a_2 - d_1), d_1 \vee (d_2 - a_1))$
$\llbracket P_1 \parallel P_2 \rrbracket_\Gamma$	$=$	$(a_1 \vee a_2, (d_1 - a_2) \vee (d_2 - a_1))$
$\llbracket P_1 \ll P_2 \rrbracket_\Gamma$	$=$	$(a_1 - d_2 \vee a_2 - d_1, d_1 \vee d_2)$
$\llbracket P_1 \parallel P_2 \rrbracket_\Gamma$	$=$	$(a_1 \wedge a_2, d_1 \wedge d_2)$
$\llbracket P_1 + P_2 \rrbracket_\Gamma$	$=$	$((a_1 \vee a_2) - (d_1 \vee d_2), (d_1 \vee d_2) - (a_1 \vee a_2))$
$\llbracket P_1 - P_2 \rrbracket_\Gamma$	$=$	$(a_1 - (a_2 \vee d_2), d_1 - (a_2 \vee d_2))$
$\llbracket P_1 \ominus P_2 \rrbracket_\Gamma$	$=$	$(a_1 \ominus (a_2 \vee d_2), d_1 \ominus (a_2 \vee d_2))$

7.3. Abbreviations

For the sake of making the presentation clear, we adopt the following abbreviations:

1. We use the terms N/A, Permit, Deny, Indeterminate(P), Indeterminate(D), Indeterminate(PD) as an abbreviation of the following situations:

$$\begin{aligned}
 \text{N/A} &= (\text{F}, \text{F}) \\
 \text{Permit} &= (\text{T}, \text{F}) \text{ or } (\text{T}, ?) \\
 \text{Deny} &= (\text{F}, \text{T}) \text{ or } (?, \text{T}) \\
 \text{Indeterminate(P)} &= (?, \text{F}) \\
 \text{Indeterminate(D)} &= (\text{F}, ?) \\
 \text{Indeterminate(PD)} &= (?, ?)
 \end{aligned}$$

2. We denote by $\varphi \rightarrow \text{p}$ and $\varphi \rightarrow \text{d}$ the following policies:

$$\begin{aligned}
 \varphi \rightarrow \text{p} &\equiv (\varphi, \perp) \\
 \varphi \rightarrow \text{d} &\equiv (\perp, \varphi)
 \end{aligned}$$

3. Any $\varphi = (d_1, \dots, d_n)$ can be represented by its restricted attributes only. An attribute a_i is restricted in φ if $d_i \neq \mathcal{D}(a_i)$. For example, if $\mathcal{A} = \langle \text{Role}, \text{Object}, \text{Action} \rangle$ such that $\mathcal{D}(\text{Role}) = \{r_1, r_2, r_3\}$, $\mathcal{D}(\text{Object}) = \{o_1, o_2, o_3, o_4\}$ and $\mathcal{D}(\text{Action}) = \{a_1, a_2\}$, then we can use the following abbreviation:

$$\begin{aligned}
 &\bullet \\
 &\quad (\{r_1, r_2, r_3\}, \{o_1\}, \{a_2\}) \\
 &\quad \equiv \\
 &\quad (\text{Object} \in \{o_1\}, \text{Action} \in \{a_2\}) \\
 &\quad \equiv \\
 &\quad (\text{Object} = o_1, \text{Action} = a_2) \\
 &\bullet \\
 &\quad (\{r_1, r_2, r_3\}, \{o_1\}, \{a_1, a_2\}) \\
 &\quad \equiv \\
 &\quad (\text{Object} \in \{o_1\}) \\
 &\quad \equiv \\
 &\quad (\text{Object} = o_1)
 \end{aligned}$$

4. The order of attributes is not important if their names are present in tuples. For example $(\text{Object} = o_1, \text{Action} = a_2)$ is the same than $(\text{Action} = a_2, \text{Object} = o_1)$. Also, we take this fact into consideration when we combine two abbreviated tuples. For instance:

$$\begin{aligned}
 &(\text{Action} = a_2, \text{Object} = o_2) \cup (\text{Object} = o_1, \text{Action} = a_1) \\
 &\equiv \\
 &(\text{Action} = a_2, \text{Object} = o_2) \cup (\text{Action} = a_1, \text{Object} = o_1) \\
 &\equiv \\
 &(\text{Action} = a_1 \text{ or } \text{Action} = a_2, \text{Object} = o_1 \text{ or } \text{Object} = o_2)
 \end{aligned}$$

Notice also that $()$ is an abbreviation of $(\mathcal{D}(\text{Role}), \mathcal{D}(\text{Object}), \mathcal{D}(\text{Action}))$ is the list of attributes is $\mathcal{A} = \langle \text{Role}, \text{Object}, \text{Action} \rangle$. In this case, it follows:

$$\begin{aligned} & (\text{Action} = a_2, \text{Object} = o_2) \cup () \\ \equiv & \\ & () \end{aligned}$$

and

$$\begin{aligned} & (\text{Action} = a_2, \text{Object} = o_2) \cap () \\ \equiv & \\ & (\text{Action} = a_2, \text{Object} = o_2) \end{aligned}$$

5. If op is a binary SePL operator, we denote by $op(P_1, P_2)$ the prefix notation of $P_1 op P_2$. For example, $P_1 \parallel P_2$ can be denoted by $\parallel(P_1, P_2)$. The result can be generalized to n compositions since all the operators of SePL are transitive. For example, $P_1 \parallel P_2 \parallel \dots \parallel P_n$ can be represented by $\parallel(P_1, P_2, \dots, P_n)$.
6. The policy $\phi : P$ is the abbreviation defined inductively as follows:

$$\begin{aligned} \phi : (\phi_1, \phi_2) &= (\phi \cap \phi_1, \phi \cap \phi_2) \\ \phi : \neg P &= \neg(\bar{\phi} : P) \\ \phi : \ulcorner P &= \ulcorner(\phi : P) \\ \phi : (P_1 \parallel P_2) &= (\phi : P_1) \parallel (\phi : P_2) \\ \phi : (P_1 + P_2) &= (\phi : P_1) + (\phi : P_2) \\ \phi : (P_1 \ominus P_2) &= (\phi : P_1) \ominus (\phi : P_2) \end{aligned}$$

where $\bar{\phi}$ is the tuple that is complementary to ϕ . For example, if $\mathcal{A} = \langle \text{Role}, \text{Object}, \text{Action} \rangle$ such that $\mathcal{D}(\text{Role}) = \{r_1, r_2, r_3\}$, $\mathcal{D}(\text{Object}) = \{o_1, o_2, o_3, o_4\}$ and $\mathcal{D}(\text{Action}) = \{a_1, a_2\}$, then

$$\overline{(\text{Action} = a_2, \text{Object} = o_2)} = (\text{Action} \in \{a_1, a_3\}, \text{Object} \in \{o_1, o_3\}, \text{Role} \in \{\})$$

7.4. From XACML to SePL

Table 7.4 outlines a BNF grammar that we propose to capture a significant subset of XACML-3.0. The literature does not contain such grammar and it is not simple to build it from XACML-3.0 specification. This will be useful also to automatically build a lexical analyzer and parser for XACML-3.0 using tools such as Lex and Yacc.

The formalization for rules and policy combining algorithms are as follows:

- **Permit-override:** Permit-overrides between P_1, \dots, P_n , denoted by $POR(P_1, \dots, P_n)$: It accepts if at least one policy accepts and denies if no one accept and at least one denies. It can be formalized in SePL as follows:

$$POR(P_1, \dots, P_n) \approx P_1 \parallel \dots \parallel P_n$$

Table 11: A BNF Grammar for a Subset of XACML-3.0

<i>PDPpolicies</i>	::=	<i>PolicySet</i> <i>Policy</i>
<i>PolicySet</i>	::=	< POLICYSET <i>Pheader</i> > [<i>Description</i>] <i>Targets</i> <i>Policies</i> [<i>Obligation</i>] [<i>Advice</i>] </ POLICYSET >
<i>Policy</i>	::=	< POLICY <i>Rheader</i> > [<i>Description</i>] <i>Targets</i> <i>Rules</i> [<i>Obligation</i>] [<i>Advice</i>] </ POLICY >
<i>Policies</i>	::=	<i>Policy</i> <i>Policy</i> <i>Policies</i>
<i>Rules</i>	::=	< RULE <i>Rheader</i> > [<i>Description</i>] [<i>Targets</i>] [<i>Condition</i>] [<i>Obligation</i>] [<i>Advice</i>] </ RULE >
<i>PShheader</i>	::=	PolicySetId = <i>string</i> Version = <i>number</i> PolicyCombiningAlgId = <i>Palg</i>
<i>Pheader</i>	::=	PolicyId = <i>string</i> Version = <i>number</i> RuleCombiningAlgId = <i>Ralg</i>
<i>Rheader</i>	::=	RuleId = <i>string</i> Effect = <i>REffect</i>
<i>Palg</i>	::=	only-one-applicable <i>Ralg</i>
<i>Ralg</i>	::=	deny-overrides permit-overrides first-applicable ordered-permit-overrides
<i>REffect</i>	::=	Permit Deny
<i>Targets</i>	::=	< TARGET > [<i>MatchAny</i>] < / TARGET >
<i>MatchAny</i>	::=	< AnyOf > <i>matchAll</i> < / AnyOf > < AnyOf > <i>matchAll</i> < / AnyOf > <i>MatchAny</i>
<i>MatchAll</i>	::=	< AllOf > <i>Matches</i> < / AnyOf > < AnyOf > <i>Matches</i> < / AllOf > <i>MatchAll</i>
<i>Matches</i>	::=	<i>Match</i> <i>Match</i> <i>Matches</i>
<i>Match</i>	::=	< Match MatchID = <i>MatchId</i> > < AttrValue > <i>value</i> < / AttrValue > < AttributeDesignator <i>ADHeader</i> / > < / Match >
<i>MatchId</i>	::=	string-equal integer-equal string-regexp-match integer-greater-than ...
<i>ADHeader</i>	::=	Category = <i>Subject</i> AttributeId = <i>AttSubject</i> DataType = <i>type</i> MustBePresent = <i>boolean</i> Category = <i>resource</i> AttributeId = <i>AttResource</i> DataType = <i>type</i> MustBePresent = <i>boolean</i> Category = <i>action</i> AttributeId = <i>AttAction</i> DataType = <i>type</i> MustBePresent = <i>boolean</i> Category = <i>environment</i> AttributeId = <i>AttEnv</i> DataType = <i>type</i> MustBePresent = <i>boolean</i>
<i>Subject</i>	::=	access-subject recipient-subject intermediary-subject ...
<i>AttSubject</i>	::=	subject-id subject-id-qualifier key-info authentication-time ...
<i>AttResource</i>	::=	resource-id target-namespace
<i>AttAction</i>	::=	action-id implied-action action-namespace
<i>AttEnv</i>	::=	current-time current-date current-dateTime
<i>type</i>	::=	x500Name rfc822Name ipAddress dnsName xpathExpression string boolean double time date dateTime anyURI hexBinary base64Binary
<i>Condition</i>	::=	< Condition > <i>BooleanExpression</i> < / Condition >

- **Deny-overrides:** Deny-overrides between P_1, \dots, P_n , denoted by $DOR(P_1, \dots, P_n)$: It denies if at least one policy denies and accepts if no one denies and at least one accepts. It can be formalized in SePL as follows:

$$DOR(P_1, \dots, P_n) \approx P_1 \parallel \dots \parallel P_n$$

- **First-Applicable:** First-Applicable between P_1, \dots, P_n , denoted by $FA(P_1, \dots, P_n)$: It accepts if there is at least one policy that accepts and that is not proceeded by a denying one and vice-versa. It can be formalized in SePL as follows:

$$FA(P_1, \dots, P_n) \approx P_1 \dots P_n$$

- **Only-one-applicable:** Only-one-applicable between P_1, \dots, P_n , denoted by $OOA(P_1, \dots, P_n)$: if more than one policy is applicable, the result will be neither accept nor deny (without decision). Otherwise, the unique applicable policy will be applied. It is formalized in SePL as follows:

$$OOA(P_1, \dots, P_n) \approx (P_1 \ominus \Sigma_{i=2}^n P_i) + \dots + (P_j \ominus \Sigma_{i=1, i \neq j}^n P_i) + \dots + (P_n \ominus \Sigma_{i=1}^{n-1} P_i)$$

- **Deny-unless-permit:** Deny-unless-permit between P_1, \dots, P_n is formalized in SePL as follows:

$$DUP(P_1, \dots, P_n) \approx \lceil (P_1 \parallel \dots \parallel P_n) \rceil$$

- **Permit-unless-deny :** Permit-unless-deny between P_1, \dots, P_n is formalized in SePL as follows:

$$DUP(P_1, \dots, P_n) \approx \lceil (P_1 \parallel \dots \parallel P_n) \rceil$$

Now, the transformation function $\lceil - \rceil$ from XACML to SePL can be inductively defined as shown in Table 7.4. Its definition is based on our own understanding of the XACML-3.0 textual description. Each rule is dedicated to a specific component (PolicySet, Policy, Rules, etc.) of XACML-3.0, where bold terms denote terminal words. For example, transforming an XACML condition (described by "< **Condition** > *BooleanExpression* < / **Condition** >") to SePL turns to extract the *BooleanExpression* and ignore the rest.

Table 12: From XACML-3.0 to SePL

	$[-] : XACML - 3.0 \rightarrow PSL$
(PolicySet)	$[\langle \text{POLICYSET } Pheader \rangle [Description] Targets Policies [Obligation] [Advice] \langle / \text{POLICYSET} \rangle]$ $= [Targets] : [Pheader] ([Policies])$
(Policy)	$[\langle \text{POLICY } Rheader \rangle [Description] Targets Rules [Obligation] [Advice] \langle / \text{POLICY} \rangle]$ $= [Targets] : [Rheader] ([Rules])$
(Policies)	$[Policy Policies] = [Policy], [Policies]$
(Rules)	$[\langle \text{RULE } Rheader \rangle [Description] [Targets] [Condition] [Obligation] [Advice] \langle / \text{RULE} \rangle]$ $= ([[Targets]] [[Condition]]) [Rheader]$
PSheader	$[PolicySetId = string \text{ Version} = number \text{ PolicyCombiningAlgId} = Palg] = [Palg]$
(Pheader)	$[PolicyId = string \text{ Version} = number \text{ RuleCombiningAlgId} = Ralg] = [Ralg]$
(Rheader)	$[RuleId = string \text{ Effect} = REffect] \Rightarrow [REffect]$
(Palg)	$[\text{only-one-applicable}] = \text{OOA}$ $[\text{deny-overrides}] = \text{DO}$ $[\text{permit-overrides}] = \text{PO}$ $[\text{first-applicable}] = \text{FA}$ $[\text{ordered-permit-overrides}] = \text{OPO}$
(REffect)	$[\text{Permit}] = p$ $[\text{Deny}] = d$
(Targets)	$[\langle \text{TARGET} \rangle [MatchAny] \langle / \text{TARGET} \rangle] = [[MatchAny]]$
(MatchAny)	$[\langle \text{AnyOf} \rangle matchAll \langle / \text{AnyOf} \rangle] = [matchAll]$ $[\langle \text{AnyOf} \rangle matchAll \langle / \text{AnyOf} \rangle MatchAny] = [matchAll] \cup [MatchAny]$
(MatchAll)	$[\langle \text{AllOf} \rangle Matches \langle / \text{AnyOf} \rangle] = [Matches]$ $[\langle \text{AnyOf} \rangle Matches \langle / \text{AllOf} \rangle MatchAll] = [Matches] \cup [MatchAll]$
(Matches)	$[Match Matches] = [Match] \cup [Matches]$
(Match)	$[\langle \text{Match MatchID} = MatchId \rangle$ $\quad \langle \text{AttrValue} \rangle value \langle / \text{AttrValue} \rangle$ $\quad \langle \text{AttributeDesignator} ADHeader / \rangle$ $\quad \langle / \text{Match} \rangle]$ $= (MatchId([ADHeader], value))$
(ADHeader)	$[\text{Category} = Subject \text{ AttributeId} = AttSubject \text{ DataType} = type \text{ MustBePresent} = boolean] = AttSubject$ $[\text{Category} = resource \text{ AttributeId} = AttResource \text{ DataType} = type \text{ MustBePresent} = boolean] = AttResource$ $[\text{Category} = action \text{ AttributeId} = AttAction \text{ DataType} = type \text{ MustBePresent} = boolean] = AttAction$ $[\text{Category} = environment \text{ AttributeId} = AttEnv \text{ DataType} = type \text{ MustBePresent} = boolean] = AttEnv$
(Condition)	$[\langle \text{Condition} \rangle BooleanExpression \langle / \text{Condition} \rangle] = BooleanExpression$